Laboratorul 7

Controlul versiunilor

1 Programe si versiuni

In cadrul ciclului uzual de dezvoltare a programelor apare adesea nevoia de a gestiona mai multe versiuni ale aceluiasi program. Fie ca e vorba de incercarea de a schimba o varianta stabila a unui program, incercare care intr-un fel sau altul esueaza si e nevoie sa revenim la varianta stabila, fie ca se doreste pastrarea ambelor variante, apare necesitatea de a gestiona versiuni multiple ale aceluiasi program de-a lungul evolutiei sale.

Acest proces, cunoscut sub numele de *Controlul Versiunilor* sau *Revision Control* (caz in care versiunile se numesc *revizii*), revine la gestiunea mai multor versiuni ale aceluiasi fisier (sau in general a unei colectii de fisiere). O revizie poate avea semnificatii multiple, de la o versiune scoasa pe piata (un asa numit *release*) pana la rezolvarea unui defect (*bug fixing*) sau adaugarea de functionalitati.

Procesul are deopotriva avantaje si dezavantaje. Avantajele tin de posibilitatea de a naviga printre revizii inainte si inapoi, de a identifica mai repede momentul aparitiei unui defect, lucrul in echipa, samd. Dezavantajele sunt legate in principal de complexitatea sporita a procesului care necesita lucru suplimentar programarii propriu-zise si organizare suplimentara a dezvoltarii programelor.

Notiunea centrala in Revision Control este *repository*-ul. Acesta este colectia tuturor versiunilor fisierelor si directoarelor dintr-un proiect. Uzual, un repository este stocat pe un server de unde utilizatorii il pot accesa. Modul lor de lucru uzual implica crearea unei copii locale a repository-ului (operatie numita *checkout*), modificarea unuia sau a mai multor fisiere si transmiterea modificarilor catre server (operatia de *commit*). Pe cale de consecinta, serverul creeaza o noua versiune a proiectului.

Aceasta secventa tipica de utilizare a repository-urilor poate fi mai complicata atunci cand repository-ul este folosit de mai multi programatori care lucreaza in echipa. Modificarile facute de unul dintre membrii echipei pot fi vazute de ceilati membri atunci cand isi actualizeaza copia locala a repository-ului prin operatia de *update*. Cu ocazia unui update se intampla adesea ca modi-

ficarile operate sa genereze conflicte. Intr-o prima faza, acestea sunt rezolvate automat, iar daca rezolutia nu este posibila, se recurge la interventia manuala a utilizatorului.

Situatii si mai complicate se inregistreaza atunci cand apar versiuni complet noi in repository care deviaza de la ramura principala de dezvoltare a proiectului numita si *trunk*. Aceste bifurcatii se numesc *ramuri* sau *branch*-uri. Integrarea branch-urilor implica o procedura de *merge* care poate genera conflicte, caz in care se aplica procedura de rezolutie a conflictelor, sau nu, caz in care merge-ul reprezinta pur si simplu o noua modificare in program.

2 Comenzi

În acest laborator vom folosi `git(1)` pentru a invata lucrul cu controlul versiunilor. Pentru a inițializa un repository nou folosiți subcomanda `init`

```
$ mkdir testrepo
$ cd testrepo
$ git init
```

Toate datele necesare funcționării repository-ului se găsesc în directorul `.git`. Un prim pas este să descrieți scopul noului proiect

```
$ vi .git/description
```

și să vă adăugați datele personale care vor fi folosite când faceți un commit.

```
$ git config user.name "Alex Alexandrescu"
$ git config user.email "alex@gmail.com"
```

Dacă doriți ca aceste date să fie folosite pentru fiecare repository de pe calculator adăugați opțiunea `--global` după `config`.

Conținutul nou, fișiere și directoare, trebuie întâi adăugat în lista de fișiere urmărite de repository cu subcomanda `add`

```
$ git add myfile.c
```

și pe urmă făcut commit cu tot ce s-a schimbat (inclusiv noile fișiere adăugate)

```
$ git commit myfile.c
```

Această comandă va porni editorul pentru a completa o descriere a schimbărilor aduse de acest commit. Completați, salvați, și ieșiți din editor. Commit-ul este efectuat.

Pentru a trimite schimbările către alte repository-uri se folosește subcomanda `push`. Întâi trebuie adăugată o intrare pentru fiecare repository cu care vrem să comunicăm folosind subcomanda `remote`

```
$ git remote add origin https://github.com/user/repo.git
```

În exemplu, `origin` este alias-ul repository-ului. Tradițional acest nume este rezervat `remote`-ului către care se va face implicit `push`

```
$ git push origin
```

Pentru a prelua schimbările de la un alt repository se folosește similar subcomanda `pull`.

```
$ git pull origin
```

Este recomandat pentru a evita pe cât posibil operațiile de *merge* să se folosească opțiunea `--rebase` care va evita un commit de tip *merge* dacă este posibil și nu apar conflicte

```
$ git pull ––rebase origin
```

Pentru a copia (sau clona) un repository existent folosiți subcomanda `clone`

```
$ git clone https://github.com/user/repo.git
```

Protocolul de comunicare poate fi `git`, `ssh`, `http` sau `ftp` în funcție de portul ales de gazdă. Implicit se folosește `ssh`.

Pentru a crea o ramură nouă se folosește subcomanda `branch`.

```
$ git branch newtopic
```

Ramura principală se numește `master` în `git(1)`. Pentru a schimba ramura folosiți comanda `checkout`

```
$ git checkout newtopic
$ git checkout master
```

Alte subcomenzi uzuale:

- `diff` – arată modificările necomise în format `diff(1)`
- `log` – arată jurnalul commit-urilor
- `show` *commit-id* – arată descrierea și schimbările aduse de un commit

3 Sarcini de laborator

1. Creați-vă un cont pe Github și adăugați-vă cheia publică (Settings → SSH and GPG keys).
2. Creați un repository nou pe github numit `hosts` și clonați-l local. Local configurați-vă numele și adresa de mail și scrieți o descriere scurtă a repository-ului.
3. În repository-ul local `hosts` creați două commit-uri: primul cu scriptul care verifica validitatea adreselor de IP din `/etc/hosts` pe care l-ati dezvoltat in laboratorul trecut. Al doilea commit il faceti cu varianta modificata a scriptului respectiv care foloseste o functie shell pentru verificarea validitatii unei adrese de IP. Functia primeste ca parametri un nume de host si o adresa de IP si verifica asocierea folosind un server DNS furnizat ca al treilea parametru al functiei.

4. Trimiteți schimbările făcute în repository-ul local către cel de pe Github.

5. Grupați-vă doi câte doi: studentul `A` împreună cu studentul `B`. `A` face un fork al repository-ului lui `B` de pe Github. în acest nou repository A face un nou commit (ex. adaugă o linie în plus care afișează numele studentului `A`) după care tot `A` face un *pull request* către `B`. `B` acceptă această cerere. Cum arată cele două repository-uri acum?